

Cosmological Simulations on a Grid of Computers

Benjamin Depardon*, Eddy Caron*, Frédéric Desprez*, Jérémy Blaizot[†] and Hélène Courtois**

*University of Lyon; ENS-Lyon/INRIA/CNRS/UCBL; LIP Laboratory. France.

[†]University of Lyon; UCB Lyon 1/CNRS/INSU; CRAL. France.

**University of Lyon; UCB Lyon 1/CNRS/IN2P3/INSU; IPN Lyon. France.

Abstract.

The work presented in this paper aims at restricting the input parameter values of the semi-analytical model used in GALICS and MOMAF, so as to derive which parameters influence the most the results, *e.g.*, star formation, feedback and halo recycling efficiencies, *etc.* Our approach is to proceed empirically: we run lots of simulations and derive the correct ranges of values. The computation time needed is so large, that we need to run on a grid of computers. Hence, we model GALICS and MOMAF execution time and output files size, and run the simulation using a grid middleware: DIET. All the complexity of accessing resources, scheduling simulations and managing data is harnessed by DIET and hidden behind a web portal accessible to the users.

Keywords: Cosmology, N-body simulations, Parallel computing, Local universe.

PACS: 98.80.Es, 97.75.Pq

1. INTRODUCTION

Cosmological simulations in this context are used to simulate the evolution of dark matter through cosmic time in various universes. A classical simulation begins with an N-body computation using for example RAMSES [?] or GADGET [?]. The output of the simulation is then post-processed using semi-analytical models such as GALICS [?] (**GALaxies In Cosmological Simulations**), then mock catalogs of observed galaxies are produced using MOMAF [?] (**Mock Map Facility**). Those models use as input a set of parameters, which influence the results, as for example: the galaxy luminosity function, the number of galaxies per observed cones, and more globally the history of star formation rate in the galaxy evolution process. Those parameters have a large range of values, the experiment here aims at reducing for each parameter the intervals of values to a subset of ranges within which the output simulations would be coherent with observed galaxies distributions.

In order to stress how realistic the post-processing results are, a first important step is to identify the GALICS and MOMAF parameters which have the largest impact on the astrophysical results, such as star formation efficiency, feedback efficiency, halo recycling efficiency. As the parameters have a large range of possible values, exploring “all” combinations is really time consuming and requires lots of computing power. To harness the difficulty of running these analysis, one need firstly to have access to many computing resources, and secondly an efficient way to access those latter. Hence, we propose a client/server implementation for running post-processings on a distributed platform composed of heterogeneous machines: a *Grid*. A transparent access to these machines is provided by a *grid middleware*: DIET (**Distributed Interactive Engineering Toolbox**). DIET handles in one common and effective way the deployment of the computations on a grid of heterogeneous and distributed computers, the management of the different components of the post-processing, and also provides monitoring, communications and computation scheduling, and data and workflows management.

Running efficiently the post-processings on a set of distributed and heterogeneous machines requires estimations on both the execution time and the amount of data to be transfered for each applications. Hence, we benchmarked GALICS and MOMAF and derived their execution time and output file size. Those models are used within DIET to select which computer should run the post-processing.

We first present in Section 2 the cosmological simulation post-processing workflow, then we give the execution time and output file size models in Section 3. In Section 4 we give an overview of the DIET middleware: its architecture, and the various features used within this project. Finally, and before concluding the paper, we present in Section 5 the

client/server implementation that allows the transparent execution of the post-processing workflow.

2. COSMOLOGICAL SIMULATIONS POST-PROCESSING

Post processing cosmological simulations is done in two steps: we first need to run GALAXYMAKER, and then MOMAF. As there isn't only a single input file, and as the intermediary files are numerous and used several times, *i.e.*, the output of GALAXYMAKER can be processed by possibly many MOMAF instances, we represent the whole execution by a workflow, *i.e.*, a graph depicting dependencies between the different tasks.

Figure 1 presents the workflows' pattern. The input of each GALAXYMAKER is a treefile (*i.e.*, a file containing the merger trees of the halos of dark matter) and a set of astrophysical parameters. In order to parallelize the hierarchical galaxy formation computation, this file can be divided into several smaller files, fed to numerous GALAXYMAKER instances. Then, all GALAXYMAKER's outputs (*i.e.*, list of galaxies) have to be post-processed by MOMAF in order to produce mock catalogs of observed galaxies.

In order to explore the parameter space, and provide different views for the virtual observations, both GALAXYMAKER and MOMAF need to be run with different sets of parameters. We have the following variable parameters:

GALAXYMAKER *variable parameters*.

- star formation efficiency (*alphapar*)
- feedback efficiency (*epsilon*)
- halo recycling efficiency (*upsilon*)
- SN feedback model (*silkfeedback*: *true* = Silk 01, *false* = SP99)

MOMAF *variable parameters*.

- the opening angle in the right ascension and declination directions (respectively *ra_size*: from 0 to 360 degrees, and *dec_size*: from 0 to 179.9 degrees).
- the minimum and maximum comoving distance (comoving distance is the distance between two points measured along a path defined at the present cosmological time) to observer, *i.e.*, all objects outside this interval is excluded (respectively *min_depth* and *max_depth*)

A given set of parameters for GALAXYMAKER produces one workflow: each treefiles are processed by an instance of GALAXYMAKER, each instance having the same set of parameters. However, different parameter sets can be fed to MOMAF within a same workflow: all GALAXYMAKER's outputs are fed to different instances of MOMAF which will produce different results. Hence, when using different parameter sets for both GALAXYMAKER and MOMAF, we obtain what is depicted on Figure 1, we have parameter sweep at two levels: within each workflow, and to generate several instances of the workflows.

3. MODELING GALAXYMAKER AND MOMAF

Having as much knowledge as possible on an application always helps running it efficiently on a distributed platform. As said previously, these applications are both computing and data intensive, and as the goal of this work is to run cosmological simulations in a parameter sweep manner, we studied the impact of each parameter on the execution time, and on the output files' size for both GALAXYMAKER and MOMAF. Benchmarks were conducted on the Grid'5000 experimental platform [?], on one of the Lyon cluster: each node has an AMD Opteron 250 CPU at 2.4GHz, with 1MB of cache and 2GB of memory.

3.1. GALAXYMAKER

We ran GALAXYMAKER on input files containing the tree files from a 512^3 particles $100Mpc.h^{-1}$ simulation. The variable parameters were: the input file size, alphapar, epsilon, upsilon and silk feedback.

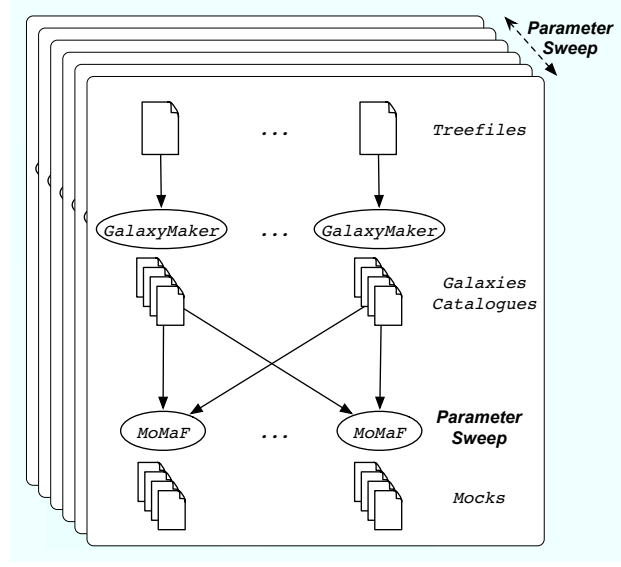


FIGURE 1. Workflows' pattern. A workflow can be executed many times on several parameter sets.

Equation (1) presents the execution time model for GALAXYMAKER. Figure 2 presents the ratio between the execution time given by the model, and the real execution time. As can be seen, for small input files, the model overestimates the execution time, this is often the case when modeling application behavior on small inputs: there is less swapping and caching problems.

$$T_{\text{GALAXYMAKER}} = a_t \times nb_{\text{halos}} + b_t + c_t \times \left(\frac{\text{alphapar}}{\text{epsilon}} \right)^{d_t} \quad (1)$$

Where T is the execution time in seconds, nb_{halos} is the number of halos found in the input file (*i.e.*, this is related to its size), a_t and b_t are constants, and c_t and d_t are linear functions of nb_{halos} . We found the following values:

- $a_t = 0.00563537$
- $b_t = -28.7845$
- if silk feedback is *false*
 - $c_t = 0.00946453 \times nb_{\text{halos}} + 71.6585$
 - $d_t = 3.64906 \cdot 10^{-7} \times nb_{\text{halos}} + 1.44597$
- if silk feedback is *true*
 - $c_t = 0.00441855 \times nb_{\text{halos}} - 40.6202$
 - $d_t = -1.83813 \cdot 10^{-7} \times nb_{\text{halos}} + 1.80095$

As can be seen, epsilon does not appear in the model, as it influences only slightly the execution time.

The output files size was easier to model: it depends only on the input file size, *i.e.*, the number of halos. Equation 2 presents the model for output file size, and Figure 3 the comparison between the model and the real output files' size. The output files' size is really stable, and thus the model perfectly matches the real output file size.

$$S_{\text{GALAXYMAKER}} = a_s \times nb_{\text{halos}} + b_s \quad (2)$$

Where S is the output files size in Mb, $a_s = 8.39317 \cdot 10^{-4}$ and $b = 9.99961 \cdot 10^{-1}$.

3.2. MoMAF

Both the execution time and the output files size of this application are less stable than GALAXYMAKER's. We weren't able to derive a model that would fit for any input file, and any parameter. Hence, we only present here the

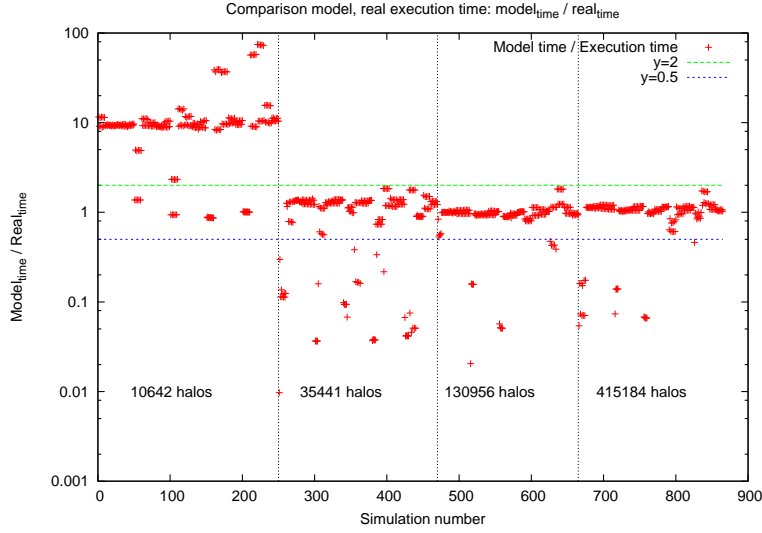


FIGURE 2. GALAXYMAKER model execution time error.

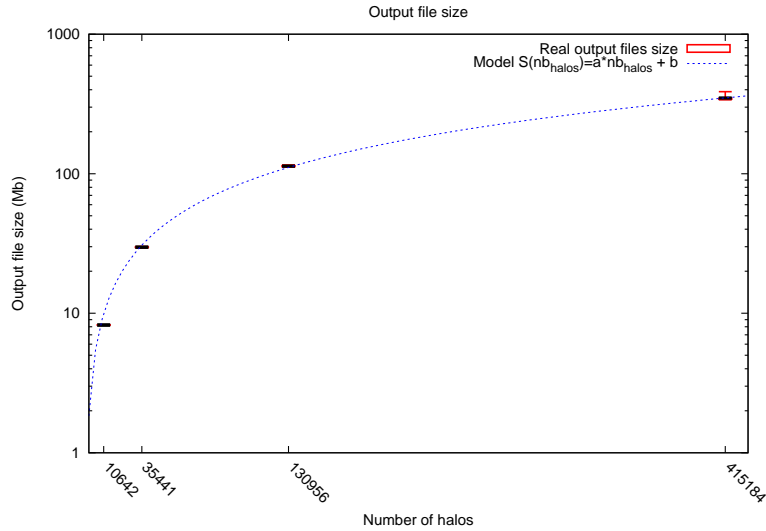


FIGURE 3. Galaxymaker output files size.

model for a given input file, and do not give the values of the various parameters as one would need to find them for each input file. Equation 3 presents the model, and Figure 4 presents the ratio between the model execution time, and the real execution time. Those were obtained by varying the following parameters: opening angles in the declination and ascension directions (*dec_size* and *ra_size*), and minimum and maximum comoving distance to observer (*min_depth* and *max_depth*).

$$T_{\text{MoMAF}} = a_t \times (\text{min_depth} - \text{max_depth})^{b_t} + c_t \quad (3)$$

With:

- $a_t = \alpha_a \times \text{ra_size}^{\beta_a} \times \text{dec_size}^{\gamma_a} + \delta_a$
- $b_t = \alpha_b \times \log(\text{ra_size}) + \beta_b \times \log(\text{dec_size}) + \gamma_b \times \text{ra_size} + \delta_b \times \text{dec_size}$
- $c_t = \alpha_c \times \text{ra_size}^{\beta_c} \times \text{dec_size}^{\gamma_c} + \delta_c$

$\alpha_{a,b,c}$, $\beta_{a,b,c}$, $\gamma_{a,b,c}$, and $\delta_{a,b,c}$ are constants.

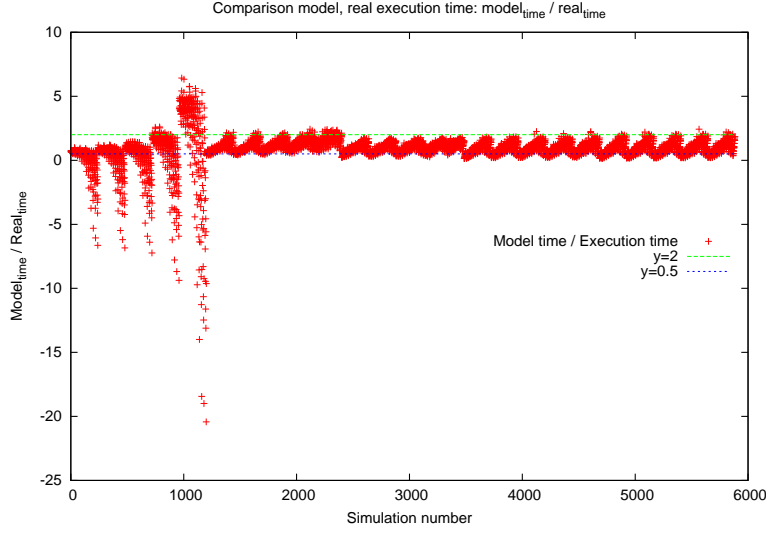


FIGURE 4. MOMAF model execution time error.

Exploring the whole parameter space, requires lots of computing power, and data storage. Those can be provided by a grid of computers, which has the advantage of having thousands of interconnected computers, but has the drawback of being highly heterogeneous and distributed on a large scale. Hence, accessing those machines is not as easy as accessing machines on a cluster. Moreover, managing thousands of workflows on a distributed environment can be really complex if merely using scripts. Thus, we need a mean of running jobs transparently on a grid, *i.e.*, a layer between the hardware and the software that hides the complexity of the platform: we need a *grid middleware*.

4. DISTRIBUTED INTERACTIVE ENGINEER TOOLBOX

We now introduce DIET, a scalable distributed middleware for accessing transparently and efficiently heterogeneous and highly distributed machines.

4.1. The DIET architecture

The DIET component architecture is structured hierarchically for improved scalability. Such an architecture is flexible and can be adapted to diverse environments, including arbitrary heterogeneous computing platforms. The DIET toolkit [? ?] (**D**istributed **I**nteractive **E**ngineering **T**oolbox) is implemented in CORBA and thus benefits from the many standardized, stable services provided by freely-available and high performance CORBA implementations. CORBA systems provide a remote method invocation facility with a high level of transparency. This transparency should not substantially affect the performance, as the communication layers in most CORBA implementations is highly optimized [?]. These factors motivate our decision to use CORBA as the communication and remote invocation fabric in DIET.

The DIET framework comprises several components. A **Client** is an application that uses the DIET infrastructure to solve problems using a remote procedure call (RPC) approach. Clients access DIET via various interfaces: web portals programmatically using published C or C++ APIs. A **SED**, or server daemon, acts as the service provider, exporting functionality via a standardized computational service interface; a single SED can offer any number of computational services. A SED can also serve as the interface and execution mechanism for either a stand-alone interactive machine or a parallel supercomputer, by interfacing with its batch scheduling facility. The third component of the DIET architecture, **agents**, facilitate the service location and invocation interactions of clients and SEDs. Collectively, a hierarchy of agents provides higher-level services such as scheduling and data management. These services are made

scalable by distributing them across a hierarchy of agents composed of a single **Master Agent (MA)** and several **Local Agents (LA)**. Figure 5 shows an example of a DIET hierarchy.

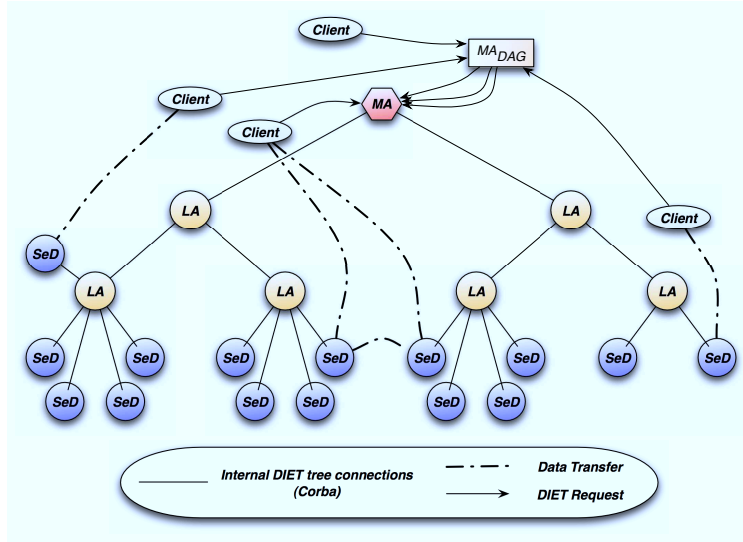


FIGURE 5. DIET hierarchical organization. Plug-in scheduler are available in each MA and LA. Clients can submit requests directly to the MA or submit workflows to the MA_{DAG} .

The **Master Agent** of a DIET hierarchy serves as the distinguished entry point from which the services contained within the hierarchy can be logically accessed. Clients identify the DIET hierarchy using a standard CORBA naming service. Clients submit requests – composed of the name of the specific computational service they require and the necessary arguments for that service – to the MA. The MA then forwards the request to its children, who subsequently forward the request to their children, such that the request is eventually received by all SEDs in the hierarchy. SEDs then evaluate their own capacity to perform the requested service; capacity can be measured in a variety of ways including an application-specific performance prediction, general server load, or local availability of datasets specifically needed by the application. SEDs forward this capacity information back up the agent hierarchy. Based on the capacities of individual SEDs to service the request at hand, agents at each level of the hierarchy reduce the set of server responses to a manageable list of server choices with the greatest potential. The server choice can be made specific to any kind of application using plug-in schedulers at each level of the hierarchy.

4.2. Data management

Data management is handled by DAGDA [?] (**D**ata **A**rrangement for **G**rid and **D**istributed **A**pplication), it allows data explicit or implicit replications and advanced data management on the grid such as data backup and restoration, persistency, data replacement algorithm. A DAGDA component is attached to each DIET element and follows the same hierarchical distribution. However, whereas DIET elements can only communicate following the hierarchy order (those communications appear when searching a service, and responding to a request), DAGDA components will use the tree to find data, but once the data is found, direct communications will be made between the owner of the data and the one which requested it. The DAGDA component associates an ID to each stored data, manages the transfers by choosing the “best” data source according to statistics about the previous transfers time and performs data research among the hierarchy. Just like DIET, DAGDA uses the CORBA interface for inter-nodes communications.

4.3. Workflows

A large number of scientific applications are represented by graphs of tasks which are connected based on their control and data dependencies. The workflow paradigm on grids is well adapted for representing such applications and the development of several workflow engines [? ? ? ?] illustrate significant and growing interest in workflow

management. Several techniques have been established in the grid community for defining workflows. The most commonly used model is the graph and especially the Directed Acyclic Graph (DAG).

DIET introduces a new kind of agent: the MA_{DAG} . This agent is connected to the MA as can be seen Figure 5. Instead of submitting requests directly to the MA, a client can submit a workflow to the MA_{DAG} , *i.e.*, an XML file containing the whole workflow description. The MA_{DAG} will then take care of the workflow execution, and schedule it along with all the other workflows present in the system, hence the system can manage multiple workflows concurrently. Thus, the client only needs to describe the workflow in an XML format, then feed in the input data, and finally retrieve the output data when the workflow has finished its execution.

4.4. Web Portal

In order to ease job submissions, a web portal has been developed¹. It hides the complexity of writing workflow XML files. The user describes astrophysical parameters, and provides the filters list file, the observation cones description file, and a tarball containing one or many treefiles, then clicks on submit, the system takes care of the communications with DIET: it creates the corresponding workflow descriptions, submit them to DIET, retrieve the final results and make them available via a webpage. In order to explore the parameter space, the submission webpage allows for astrophysical and cones parameters to define three values: the minimum and maximum values, and the increase step that needs to be applied between these two values. Thus, with the help of a single webpage, the user is able to submit lots of workflows.

In case a submission fails, the job is submitted again after a fixed period, this until the job finishes properly, or is canceled by the user. Once finished, a tarball file containing all result files can be downloaded. Depending on the option chosen at the submission stage, this file contains output files of both GALAXYMAKER and MOMAF, or just MOMAF.

Access to this system is protected by a login/password authentication, so as to restrict access to applications and produced data. This system also provides independent parameter sweep jobs submissions for both applications.

5. FRAMEWORK

5.1. Client

The main idea is to provide a transparent access to computing services. End users shouldn't have to write a single line of code, or XML to be able to use the submission system. Thus, the DIET client only needs to parameters: a tarball file containing all treefiles to process, and a file containing the parameters for both GALAXYMAKER and MOMAF. This parameter file uses the same syntax than the ones used with GALAXYMAKER and MOMAF, hence one only has to concatenate GALAXYMAKER parameter file with MOMAF parameter file and remove any double. The client in itself does not allow parameter sweep for GALAXYMAKER, one has to call the client once for each GALAXYMAKER parameter set. However, parameter sweep is provided for MOMAF. This behavior reflects the workflow description depicted on Figure 1.

The client automatically creates the workflow description XML: for each treefile a GALAXYMAKER service is added, and for each MOMAF parameter set, a MOMAF service is added. The output of all GALAXYMAKERS are fed into each single instance of MOMAF.

The workflow is then submitted to DIET. Data transfers and services execution are automatically handled. Once the execution has ended, all output data is retrieved at the client level: for each GALAXYMAKER and MOMAF output data is stored in an independent directory. If the client is called from the DIETWEBBOARD, all produced data is compressed and send back to DIETWEBBOARD's storage disk and made available on the webpage.

Cosmological simulations generate a lot of temporary files locally at the client level, as well as on every server that executed a GALAXYMAKER or MOMAF service. Thus, once everything has been executed, and useful data retrieved,

¹ Portal for cosmological simulations submission: <http://graal.ens-lyon.fr:5544/Cosmo/>
Description of the available features: <http://graal.ens-lyon.fr/DIET/dietwebboard.html>

the client calls cleaning services which deletes now obsolete data on each server, this mechanism is presented in more details in the following section.

5.2. Server

In this section we describe the mechanisms used by the servers in order to manage data, and in order to efficiently access resources.

Data management. These cosmological applications require as input large amounts of data, the outcome of the processing is also very large. Depending on the input parameters, as well as the number of filters applied in the simulation, the number of files can vary. A problem arises: how to transfer efficiently data between GALAXYMAKER, MOMAF and the client. Part of the data produced at one step need to be transmitted efficiently to the next step, while some of them need to be sent back to the client. Creating an archive by compressing produced files could be a solution to deal with so many files, however, this would lead to increasing disk usage (at least temporary, as it would require twice the disk space used by the files at a given time), as well as computation time (compressing or even storing files in an archive can be costly). The option we chose is to have all produced data added into a particular DIET data type: a container. A container can contain any kind of DIET data (be it a file, an integer, a vector, ...), and we can add as many element as needed: the size is dynamically managed. This structure allows to have an undetermined number of files handled in a single data type, and thus will be efficiently managed by DIET when data movement is required.

Data is managed by DAGDA so as to benefit from data persistency and replication. Produced data is kept locally at SED level, and is downloaded on demand by the SED executing the following service, and/or the client.

As said previously, cosmological simulations produce a lot of data that has to be kept during the whole workflow execution. Indeed, each GALAXYMAKER output has to be post-processed by possibly many MOMAF instances. A server only has local knowledge on the platform: it does not know anything about the whole workflow, nor about the other services' needs in terms of data: once a service has executed, it has no means of knowing whether a data it has produced has already been downloaded or not, and whether it will be needed in the future, hence it has to keep. As a server cannot know by no means if a workflow has ended, data would possibly stay indefinitely on the server, which is of course impossible due to disk space constraints. One way of dealing with this problem would be to let DAGDA handle data removal when the disk is full, according to a specified replacement algorithm (currently three such algorithms are available: Least Recently Used, Least Frequently Used and First In First Out). However this does not solve our problem as DAGDA could remove data still in use. To cope with this problem, a new feature has been added to DIET. Usually a service is defined by the SED programmer in a static manner, this service is instantiated when the SED is run, and ends when the SED is killed. This does not reflect the dynamic behavior of our workflow executions. Thus we added the possibility to add and remove services dynamically at runtime. Thus, whenever a service is called, the SED keeps track of all created files and directories. Just before the service ends, a cleanup service is spawned: its service name is unique for the whole DIET platform (the service name is generated using a Universally Unique Identifier, UUID, giving us reasonable confidence that the service name is unique), and it requires no parameters, as all required information is kept by the SED. When this service is called (*i.e.*, when the MA_{DAG} detects that no services need the data anymore), it deletes the corresponding previously registered files and directories. Furthermore, in order to prevent many calls to this cleaning service, the service automatically removes itself from the list of available services. Hence, the machine is left clean, with no temporary files remaining on the disk. Figure 6 depicts the data management process when executing the whole workflow.

Accessing resources. In order to be able to execute a service, a SED need to access computing resources. It has in fact two ways to do so: either the service is run locally on the machine the SED is deployed, or the SED can interact with a Batch system in order to submit a job to a cluster or a grid. In the former case the service only has access to the machine on which the SED runs, but in the latter case, a service can be run in parallel on many machines at the same. Note however that if one has control on a cluster, nothing prevents her from deploying a SED per node on the cluster, and thus DIET won't have to deal with the Batch system.

When interacting with the Batch system, the SED needs to be deployed where the Batch system runs, *e.g.*, the gateway, in order to be able to communicate with it. In this case, the services aren't executed on the gateway, but a node is requested to the batch system, *i.e.*, a script is submitted to the system. This method complicates data management, as once the script is submitted it cannot directly interact with DIET, hence data has to be retrieved first by the SED on the gateway in a local directory (or a shared directory if available), then, when the reservation is available, *i.e.*, the

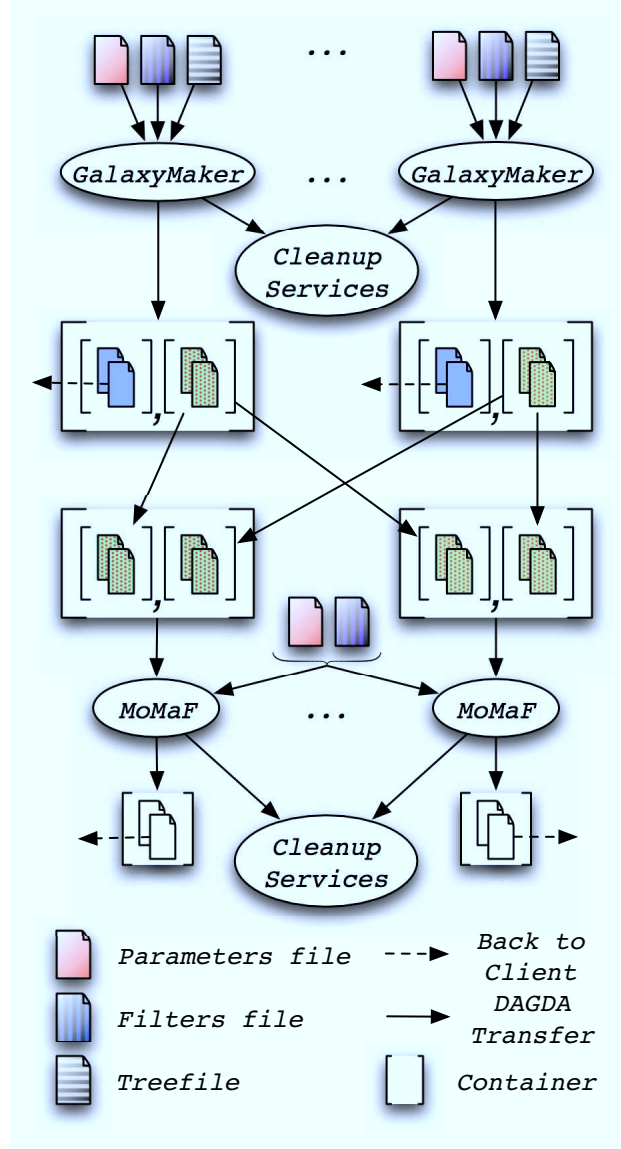


FIGURE 6. Data management.

script starts running, required data is sent to the corresponding node, where the application is executed. Finally, when the service ends, the script sends all produced data back to the gateway. The SED detects the script termination, adds produced data to DAGDA so that the client and/or the next step can retrieve it.

In order to be able to make reservations on the batch system, we need an estimation of the job running time (*i.e.*, whenever a reservation is made on a batch system, one has to specify its duration). As presented in Section 3, an execution time model was obtained via benchmarking GALAXYMAKER and MOMAF with various parameter sets. Even if these models aren't totally accurate (there may be an error factor of 2, or even greater for small data sets), they give us a rough estimation that can be used for submitting jobs to a batch system, as well as for scheduling purposes between workflows: the MA_{DAG} uses the HEFT (Heterogeneous Earliest Time First) heuristic [?] to schedule the different tasks. Hence, whenever a request is submitted, each SED estimates the execution time it would take to execute the request based on the model and the previous executions. An history is kept locally at each SED in order to dynamically correct the model parameters: at the end of each correct execution, the SED updates a frame based history (it keeps the bias between the model and the real execution time for the last n executions), and uses this history

to predict the next execution time. This technique copes with two problems: errors induced by the model, and the fact that machines may be heterogeneous and different from the ones used for the benchmarks.

6. CONCLUSION

In this paper, we presented models for GALAXYMAKER's and MOMAF's execution time, and for GALAXYMAKER's output files's size. Execution time models give an estimation that is quite close to the real execution time for input files having a reasonable size. For small input files, the model returns an overestimate of the execution time, but this is easily explained by cache mechanisms at the hardware level, and can be taken into account in the model.

We also modeled the post-processing workflow, and provided efficient means of executed it in a parameter sweep manner on a grid of computers. Our solution relies on the DIET middleware that provides transparent access to resources, and data and workflows management. All the complexity of running the post-processing on a grid, and the creation of the workflows are hidden by a web interface, that provides an easy and user-friendly way of submitting such cosmological simulations post-processing on many heterogeneous and distributed resources.

The next step is of course to analyze the results produced by the post-processing, and derive which parameters are the most influent on the results. Finally, we aim at providing ranges of values for the different parameters, that would provide correct results, *i.e.*, comparable to observational data.

ACKNOWLEDGMENTS

This work was developed with financial support from the ANR (Agence Nationale de la Recherche) through the LEGO project referenced ANR-05-CIGC-11.